第2章 機能分散型マルチコア設計を体験する

――小型のテレビ電話端末を複数のCPUコアで実現

大牧正知

本稿では，複数のCPUコアを一つのLSIに搭載するマルチコアLSIの設計の手順を解説する．サンプルとなる設計は，テレビ電話システムである．CPUコアとして，米国Xilinx社のMicroBlazeを使用する．（編集部）

複数のCPUを用いる並列処理は，昔からいろいろと研究されていますが，その多くは汎用コンピュータを対象にしたものでした．

実行するアプリケーションが限られている組み込みシステムは，並列処理にもっとも適した分野の一つです．しかし，実際に組み込みの分野でマルチCPUシステムが使われることは，あまり多くなかったと思います．その理由として，複数のCPUを搭載することによる部品や基板のコスト，および基板サイズの増大がいちばんの原因ではないでしょうか．

1チップに複数のCPUを内蔵するのであれば，基板のコストやサイズは問題となりません．LSIの規模が大きくなればコストアップにつながりそうですが，近年のプロセス微細化のおかげで，CPUコアを追加することによるコストへの影響は小さなものになっています．周辺機能のための部品は，CPUが一つであったときと変わりません．特定の小規模な処理しか行わないソフトウェアであれば，開発は比較的容易であり，必要なメモリ容量も小さくてすみます．タスク切り替えといったオーバヘッドもなくなります．ピーク性能に合わせた高性能CPUを搭載するより，機能ごとに適当な性能のCPUコアを複数使用したほうが，システム・コストを抑えられる可能性があります．

本稿では，テレビ電話端末の設計を例に，一つのFPGAに複数のCPUコアを実装する手法を解説します．具体的な実装方法を説明し，性能についても考察します．CPUコアとしては，米国Xilinx社がソフト・マクロで提供する「MicroBlaze」を用います．

1．複数のCPUによる並列処理

並列処理とひと言でいっても，一つのCPU内に実行ユニットが複数ある場合（いわゆるスーパスカラ）から，インターネットなどのような大規模ネットワークでコンピュータ間を接続する場合（いわゆるクラスタ）まで，さまざまな



図1
負荷分散と機能分散
（a）負荷分散　（b）機能分散

ものがあります．ここでは，複数のCPUを一つの装置内に持つ場合の並列処理について考えてみたいと思います．

● 機能分散と負荷分散

パソコンの世界でも，すでにマルチプロセッサのハードウェアは浸透してきています．しかし，ソフトウェアの対応のほうはまだまだこれからという感じのようです．実際，ほとんどのパソコン用アプリケーション（ソフトウェア）は，マルチプロセッサに対応していません．対応しているのはMPEGのエンコード・ソフトウェアなど，ごく一部です．しかし，だからといってマルチプロセッサがむだというわけではありません．複数のアプリケーションを同時に動かした場合，そのアプリケーションがマルチプロセッサに対応していなくても，シングル・プロセッサのパソコンと比べて，速く実行することができます．

このように，マルチプロセッサによる並列処理は，一つのアプリケーションで複数のプロセッサを使い，処理速度を上げる方法（負荷分散）と，複数のアプリケーションそれぞれにプロセッサを割り当てる方法（機能分散）の2種類に，大きく分けられます（図1）．

パソコンの例でもわかるように，負荷分散はアプリケーションがマルチプロセッサに対応していないといけないのに対し，機能分散はアプリケーションがマルチプロセッサに対応していなくても実装可能です．したがって，並列処理の実装は負荷分散に比べて，比較的容易になります．

● マルチタスクと機能分散

一つのプロセッサでいろいろな処理を同時に行いたい場合，マルチタスクOSを使って，処理をこまめに切り替えて実行させなければなりません．この場合，タスク切り替えごとにオーバヘッドが発生します．組み込み機器では，ある入力に対して決められた時間内に応答しないといけないケースがよくあります．一つのプロセッサでいろいろな処理をさせていると，ある処理中にほかの処理の要求が発生して処理が遅延するなど，いろいろと問題が起こります．このため，より高い性能のCPUが要求されることになります．

処理ごとにCPUを用意すると，このような問題はすべて解決されます．CPUは一つの処理に専念できるので，タ

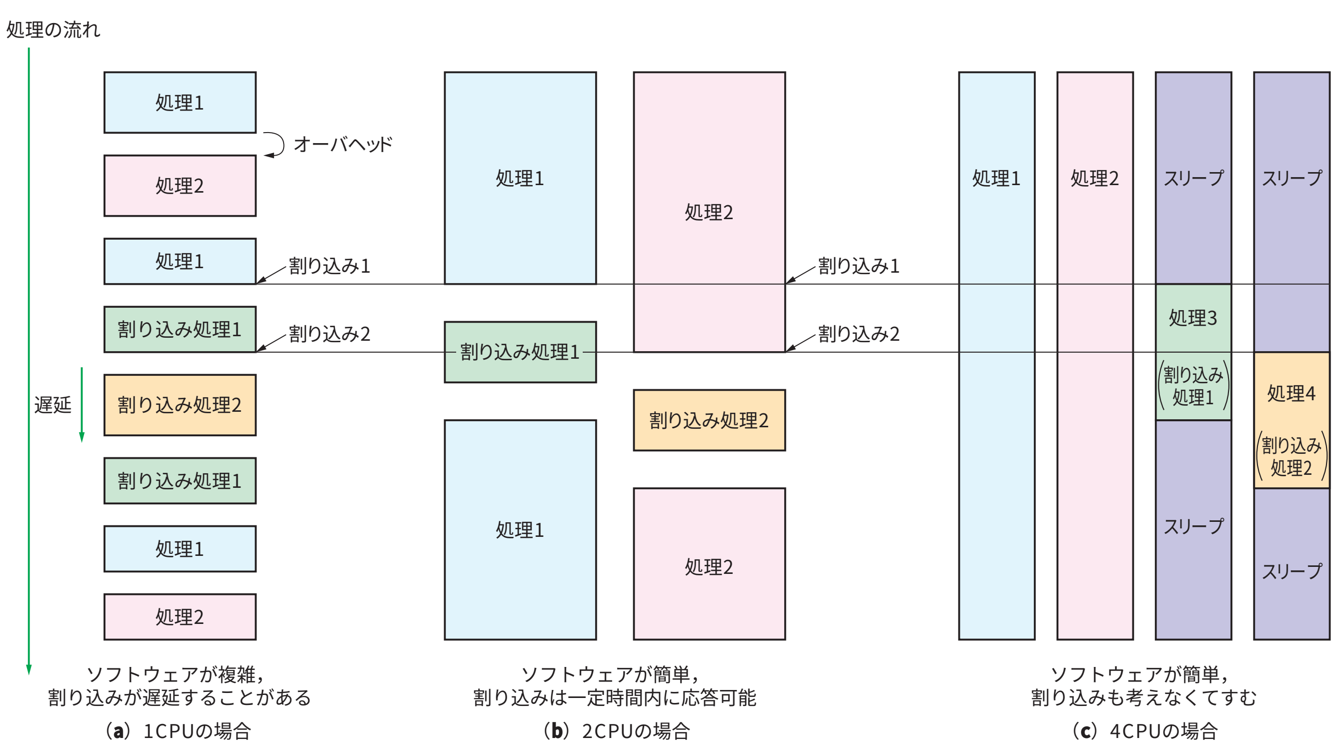


図2　マルチタスクと機能分散

二つの同時実行処理と二つの割り込み要因が存在する場合を示す．1CPUでは，一定時間ごとに処理1と処理2を切り替え，さらに多重割り込み処理も考慮しなければならない．各処理の切り替えのためのオーバヘッドも多い．2CPUでは，あらかじめ割り当てられた処理に専念できる．それぞれ割り当てられている一つの割り込み処理だけを考慮すればよい．(c)の4CPUの場合，四つの処理を各CPUに割り当ててしまえば，処理の切り替えはいっさい起こらない．一つの処理に長い時間をかけられるのであれば，要求されるCPU性能が低くなる．

スクの切り替えを考える必要がなくなります(**図2**)．割り込みやタスク切り替えを考えなくてよいのであれば，設計が非常に楽になります．マルチタスクOSは必要なくなり，オーバヘッドも発生しません．一つ一つのプロセッサに要求される性能も抑えられます．したがって，トータル・コストを抑えられる可能性があります．

このように，機能分散の並列処理は実装が楽であるうえに，全体の性能を容易に向上させることができます．

● 機能分散マルチコア・システムの構成

機能分散マルチコア・システムの場合，複数の機能ブロックの間で同期を取ったり通信したりする場合があるものの，処理そのものは完全に独立しています．そこで，さまざまなシステム構成が考えられます．

いちばんわかりやすいのは，それぞれのCPUコアがメモリや周辺I/Oを持った，独立したシステムで動かす場合です．この場合，単純にCPUコアの数に比例した性能が得られますが，メモリや周辺I/Oがそれぞれに必要になります(**図3(a)**)．

共通のバス上に配置して，周辺I/Oやメモリを共有化する構成も考えられます(**図3(b)**)．システム・コストは，CPUコアごとに独立した構成をとる場合より抑えられる可能性が高くなります．メモリ上のデータを共有できるので，データのやりとりが簡単になります．

しかし，共有メモリ上に命令コードやデータを格納すると，頻繁にメモリ・アクセスが発生します．バス・アクセスが競合し，それぞれ独立したシステムで動かす場合と比べると，性能面で影響が出るかもしれません．そのため，このようなシステムの場合は，キャッシュ・メモリや専用の通信ポートを各CPUコアが持つことで，バス・アクセスを抑えることがあります(**図3(c)**)．

また，バスを共有する場合に，各CPUにローカル・メモリを持たせる構成も考えられます．各CPUコアで動かすソフトウェアや共有する必要のないデータにはローカル・メモリを使います．こうすることで，バス・アクセスを最小にしつつ，データや周辺I/Oを共有化できます．性能を保ちつつコストを抑える構成の一つです(**図3(d)**)．

2. テレビ電話端末のシステム構成

本稿では，小型のテレビ電話端末を例に，マルチコア・システムの設計を考えてみたいと思います．

2

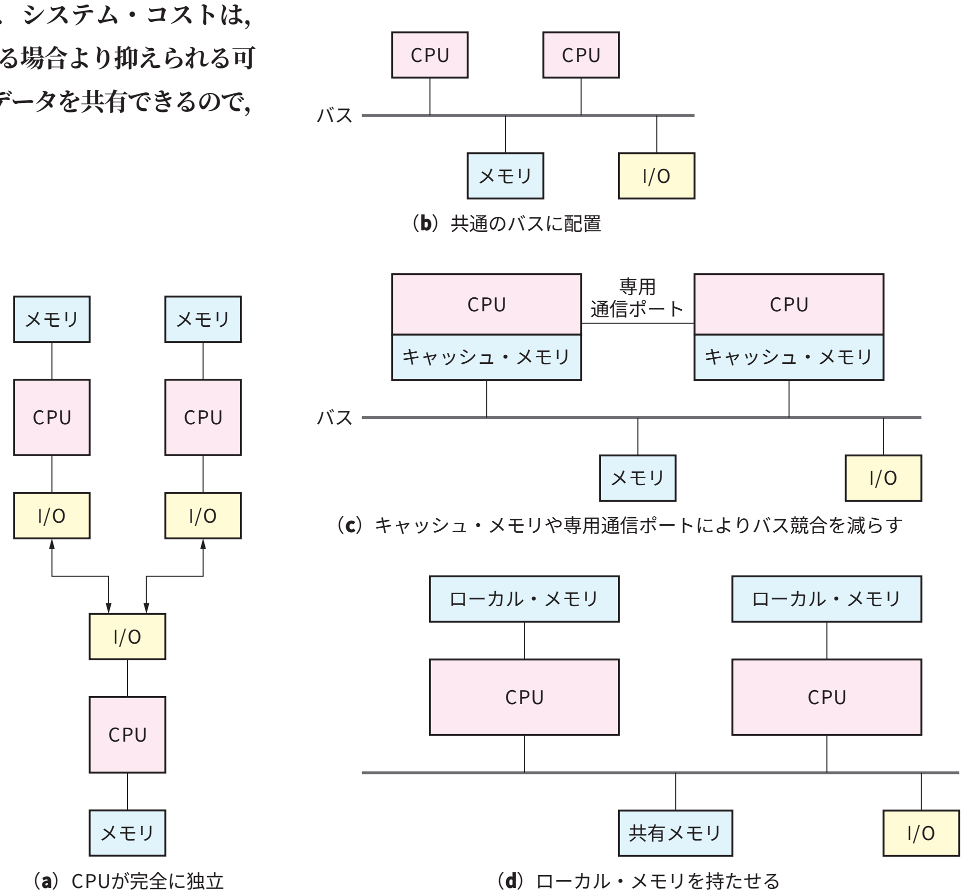


図3
機能分散マルチコア・システムの構成
それぞれのCPUコアがメモリや周辺I/Oを持った独立した構成，共通のバス上に配置して周辺I/Oやメモリを共有化した構成，キャッシュ・メモリや専用の通信ポートによりバス・アクセスを抑えるしくみを持たせた構成，各CPUにローカル・メモリを持たせた構成などが考えられる．

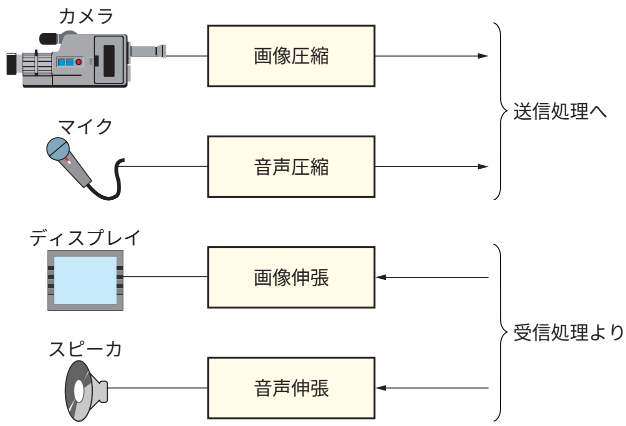


図4　テレビ電話端末のシステム構成
電話としての通信機能のほか，例えば画像圧縮/伸張，音声圧縮/伸張が必要．

テレビ電話端末で要求される処理としては，電話としての通信機能のほか，例えば，画像の圧縮/伸張，音声の圧縮/伸張が考えられます．これらを一つの装置に組み込む必要があります(**図4**)．

● ASSPはコスト増，1CPUでは機能に限界がある

すべての処理を，それぞれの機能を持ったLSI(ASSP：application specific standard product)を基板上に並べて構成することも可能かもしれません．しかし，各ASSPごとにメモリが必要になるなど，基板上の部品点数は相当多くなってしまいそうです． これは部品管理コストや基板コストの増大，装置の大型化につながります．

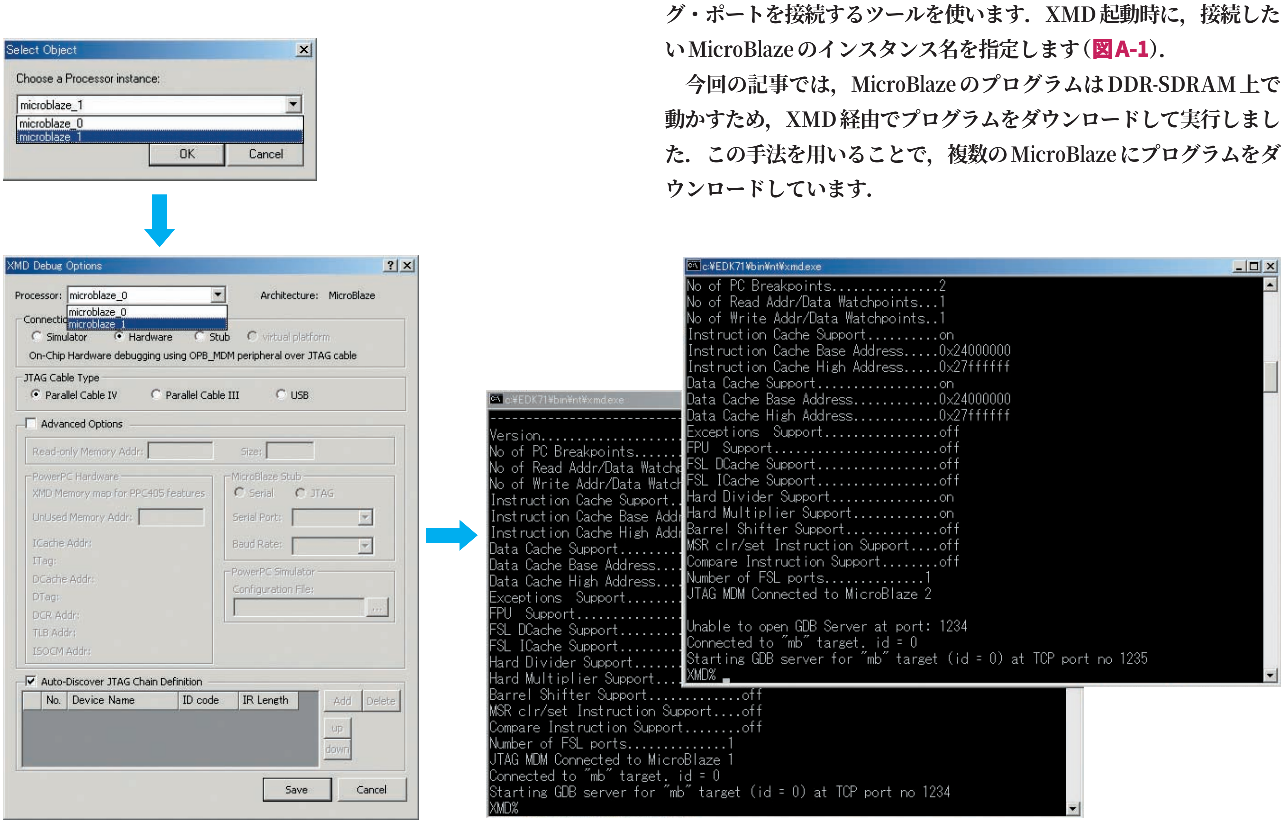
CPUで処理させるのも一つの方法です．しかし，この四

Column1　MicroBlazeにおけるマルチプロセッサ時のデバッグ

MicroBlazeでデバッグを行う際，通常，FPGA書き込みケーブルである「Pallarel Cable IV」などをFPGAのJTAGピンに接続してデバッグします．一つのFPGA内にMicroBlazeを複数個使用した場合はどうするのでしょうか．

開発キット(EDK)では，複数のMicroBlazeが同じFPGA内にあっても，CPU番号でそれぞれを識別して同時にデバッグ・ポートを接続することが可能です．接続方法は簡単で，XMDというデバッグ・ポートを接続するツールを使います．XMD起動時に，接続したいMicroBlazeのインスタンス名を指定します(**図A-1**)．

今回の記事では，MicroBlazeのプログラムはDDR-SDRAM上で動かすため，XMD経由でプログラムをダウンロードして実行しました．この手法を用いることで，複数のMicroBlazeにプログラムをダウンロードしています．



図A-1　マルチプロセッサ時のデバッグ

つの処理のすべてを一つのCPUで行うとなるとたいへんそうです．映像のほうはコマ送りのようになってしまってもよいかもしれませんが，音声は処理が遅れてタイミングがずれると，音とびを起こしてしまうので致命的です．電話としての最低限の要求を満たせません．かならず一定時間内に処理を実行する必要があります．高い性能を持つCPUとリアルタイムOSを使えば実現できそうですが，パソコンのように放熱ファンが動き続ける電話なんてありえません．こう考えると，一つのCPUによるシステム構成には限界がありそうです．

CPUで処理させるもう一つの方法として，複数のCPUを使い，機能ごとにCPUを割り当てる構成があります．この構成は，複数のCPUを使うことから一見難しそうですが，ソフトウェアまで考慮すると実は設計がシンプルになります．一つのCPUはほかからの割り込みなどの要求を受ける必要がなく，一つの機能に専念できるため，処理が遅れる可能性が少なくなります．また，多数の割り込みを扱う必要がなくなるので，ソフトウェアがシンプルになり，実装も容易になります．

● FPGA向けソフトCPU「MicroBlaze」

機能ごとにCPUを用意する場合，「ソフトCPU」と呼ばれるIPコアを活用するのが便利でしょう．例えば，Xilinx社が提供するFPGA向けソフト・マクロのCPUとして，MicroBlazeがあります(**図5**)．FPGAの論理ブロックを使って構成するCPUのIPコアです．MicroBlazeについては，本誌でも過去に何度か解説されています．詳しい説明は参考文献をご覧ください．ちなみに，Xilinx社のVirtex-II PROやVirtex-4 FXに内蔵されているPowerPCはハード・マクロのCPUです．

ソフトCPUの一つの特徴として，FPGAのリソースの許す限り，一つのFPGAの中に何個でもCPUを構成できるということがあります．2個とか4個といった少量ではなく，FPGAのリソースが許しさえすれば，16個でも32個でも構成できます．必要な数だけ使えるということは，機能ごとにCPUを割り当てることが容易であり，機能分散型のマルチコアLSI設計に適しています．

通常，CPUが動くためには，命令やデータを格納するためのメモリが必要になります．マルチCPUシステムの場合，それぞれにメモリを持たせるか，共有メモリを使うかのどちらかになります．汎用のCPUによるマルチCPUシステムでは，それぞれにメモリを持たせる場合，部品点数が増え，コスト増につながります．共有メモリを使う場合は，チップセットや内蔵のメモリ・コントローラが共有メモリに対応していなければなりません．もし対応していなければ，外部で回路を組まなければなりません．これもやはり，コストや設計工数を増やしてしまいます．

MicroBlazeは，OPB(On-chip Peripheral Bus)というオンチップ・バスをサポートしています．OPBはマルチマスタをサポートしており，複数のMicroBlazeを一つのOPBに接続することができます．同じOPBに接続されている周辺I/Oは，すべてのMicroBlazeからアクセス可能となり，メモリだけでなく，UARTやタイマなど，すべての周辺機能を共有することができます．

また，MicroBlazeは外部メモリだけでなく，FPGA内部のメモリ・ブロック(BlockRAM)もメモリとして使うことができます．小規模なソフトウェアであれば，このメモリ・ブロックに収めてしまい，外部の共有メモリを使わず

FSL：Fast Simplex Link
LMB：Local Memory Bus
OPB：On-chip Peripheral Bus

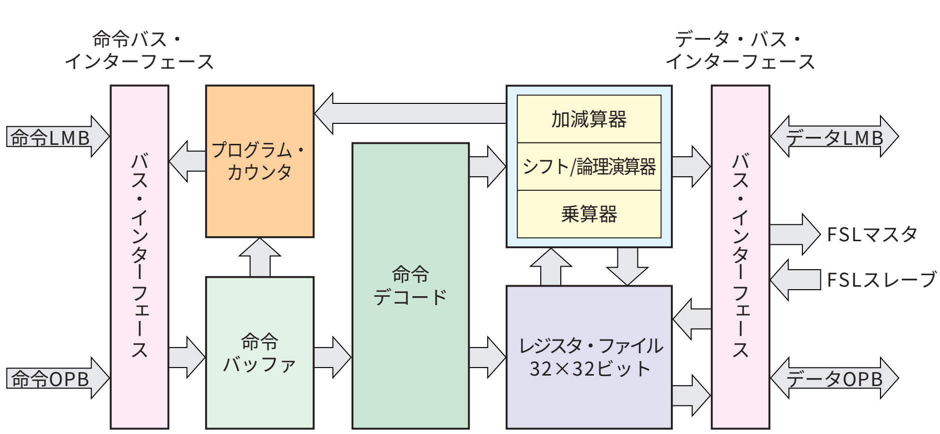


図5 MicroBlazeのブロック図
オンチップ・バス(OPB)やオンチップ・メモリとのインターフェースを持つ．

に実行させることも可能です．一般にCPUコアは，オンチップ・メモリで動作させた場合に最大の性能を発揮できます．FPGAのメモリ・ブロックは，プログラムを格納するローカル・メモリとして便利です．

さらに，MicroBlazeはFSL（Fast Simplex Link）というインターフェースを持っています．これは，アドレス・バスを持たない，データ・ポートのみのストリーム系のインターフェースです．非常に高速なうえ，データを受信するときにデータが来ていないと，データが来るまで待つブロッキングの動作モードを持ちます．プロセッサ間の動作の同期化など，プロセッサ間通信を行いやすいインターフェースになっています．

このようにMicroBlazeは並列処理のためのさまざまな機能を持っており，手軽に並列処理を実装することができます．

3. マルチコアLSIの設計フロー

MicroBlazeを使ったマルチコア・システムの設計法を具体的に説明します．今回使うボードは，米国Memec Design社のFPGAボード「Virtex-4 LC FX12 Development Board」です．FPGAとして，Xilinx社のVirtex-4 FXを搭載しています．Virtex-4 FXは，PowerPCのハード・マクロを内蔵するFPGAですが，今回はこのPowerPCを使わず，ソフト・マクロのMicroBlazeのみを使います．

● 画像圧縮/伸張にそれぞれCPUコアを割り当てる

まず，画像処理部だけを設計してみましょう．ブロック図を図6に示します．

JPEG画像圧縮用，伸張用にそれぞれ一つずつのMicroBlazeを使います．消費リソースを最小限にするために，メモリやタイマなどは共有化して使います．

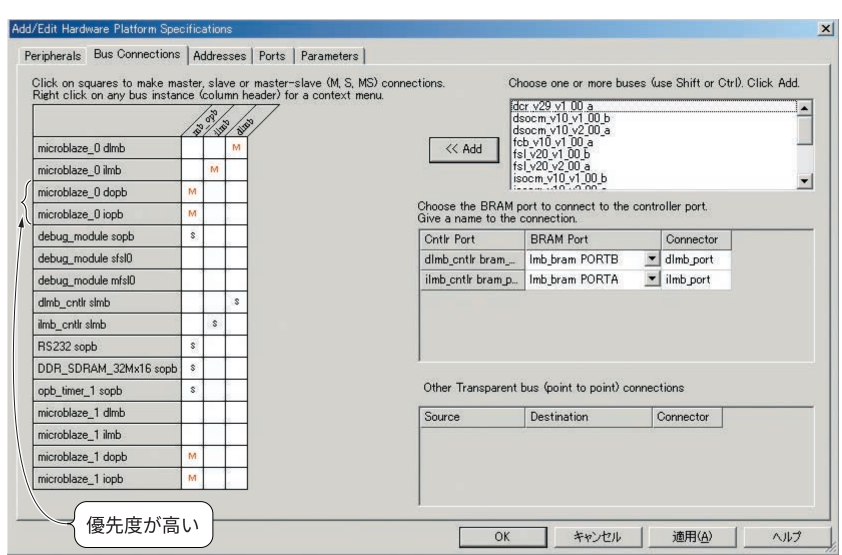
このような構成では，二つのMicroBlazeを一つのOPBに接続するだけです．MicroBlazeは命令バスとデータ・バスが独立しているため，同じOPBに接続されるマスタは四つになります．そして，バスの競合を最小限にするために，MicroBlazeにキャッシュを持たせます．

今回使うボードには，DDR SDRAMが搭載されています．開発キット（EDK）付属のDDR SDRAMコントローラは，マルチポートのものがあり，これとMicroBlazeのキ

Column2 バスの優先度

OPBは，各マスタごとに優先度があり，OPBへのアクセスが同時に発生した際には，優先度の高いほうのマスタがバスを使用することになります．

それでは，この優先度が何によって決まっているかというと，OPBへの接続順です．Add/Edit cores画面で，上のほうで接続されているマスタが，もっとも優先度の高いマスタになります．また，OPBバスのオプションを利用して，この優先度をソフトウェアから変更できるようにすることも可能です（図B-1）．



図B-1
バスの優先度

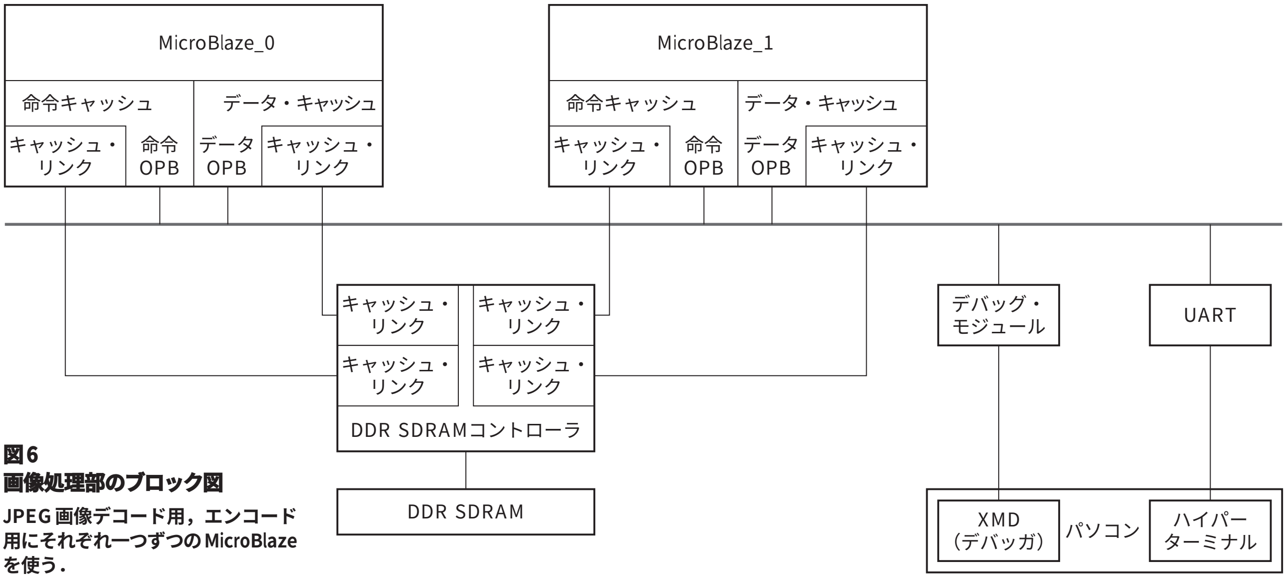


図6
画像処理部のブロック図
JPEG画像デコード用，エンコード用にそれぞれ一つずつのMicroBlazeを使う．

ャッシュ・リンク（バスを介さずに外部メモリのデータをキャッシュ・メモリに直接転送する機能）を組み合わせることにより，バースト転送が可能となって，MicroBlazeのキャッシュ性能が大幅に向上します．また，バースト・データ転送にバスを使わず，このキャッシュ・リンクを使うことになるため，バスの競合が起こりにくくなります．したがって，今回はこのキャッシュ・リンク＋マルチポートDDR SDRAMコントローラを使います．

● MicroBlaze実装の基本フロー

MicroBlazeでは，EDKに付属した機能である「Base SystemBuilder」を使って基本システムの設計を進めていきます（**図7**）．

まずは，EDKのGUIであるXPSを起動し，BaseSystem Buiderを立ち上げます（**図7**の①）．次に，プロジェクトの保存場所と，ボード定義ファイルを指定します（**図7**の②）．新規の設計を行うように指定して（**図7**の③），使用するボードを選びます（**図7**の④）．ここまでで初期設定が終わります．

次の画面から，ハードウェア構成の設定になります．まず，どのCPUを使うかを選びます．ターゲットFPGAがVirtex-4 FXの場合は，PowerPCとMicroBlazeの両方を選択可能です．今回はMicroBlazeを使います（**図7**の⑤）．次にMicroBlazeの設定を行います．MicroBlazeの動作周波数はVirtex-4ファミリであれば，100MHz程度は問題ありません．しかし，民生機器向けの設計の場合，量産時には低コストFPGAを使うのが一般的です（大量生産品ではASIC化することになるが，ここではそれは考えないことにする）．例えば，Xilinx社のFPGAであればSpartan-3ファミリを使うことを考慮しなければなりません．そこで今回は，動作周波数を66MHzにしました．動作周波数に余裕を持たせておくと，配置配線時に時間がかかったり，リソース不足で制約を満たせないといった問題も起こりにくくなり，結果的に開発期間が短縮されます．デバッグ・モジュールはデフォルトのハードウェア・デバッグにしておき，キャッシュはキャッシュ・リンクを有効にしておきます（**図7**の⑥）．

次の画面から，使う周辺回路のIPコアを選択していきます．今回は，結果のログを出力するためのUARTと，DDR SDRAMコントローラ，時間計測用のタイマを使います（**図7**の⑦～⑪）．最後に，キャッシュ・メモリ容量と，キャッシャブル空間を設定（**図7**の⑫）して，BaseSystem Builderの設定は完了です．

● 二つ目のCPUコアを追加する

BaseSystemBuilderは，MicroBlazeを一つしか構築してくれません．そこで，このでき上がった設計にさらにMicroBlazeを追加します．

CPUコアの追加は，Projectメニューの「Add/Edit Cores」からGUIベースで設定できます．しかし，あらかじめ設定したMicroBlazeとほぼ同じ構成のコアを追加するときは，ハードウェア設定情報が記述されているMHSファイルを直接編集するほうが簡単です．

2

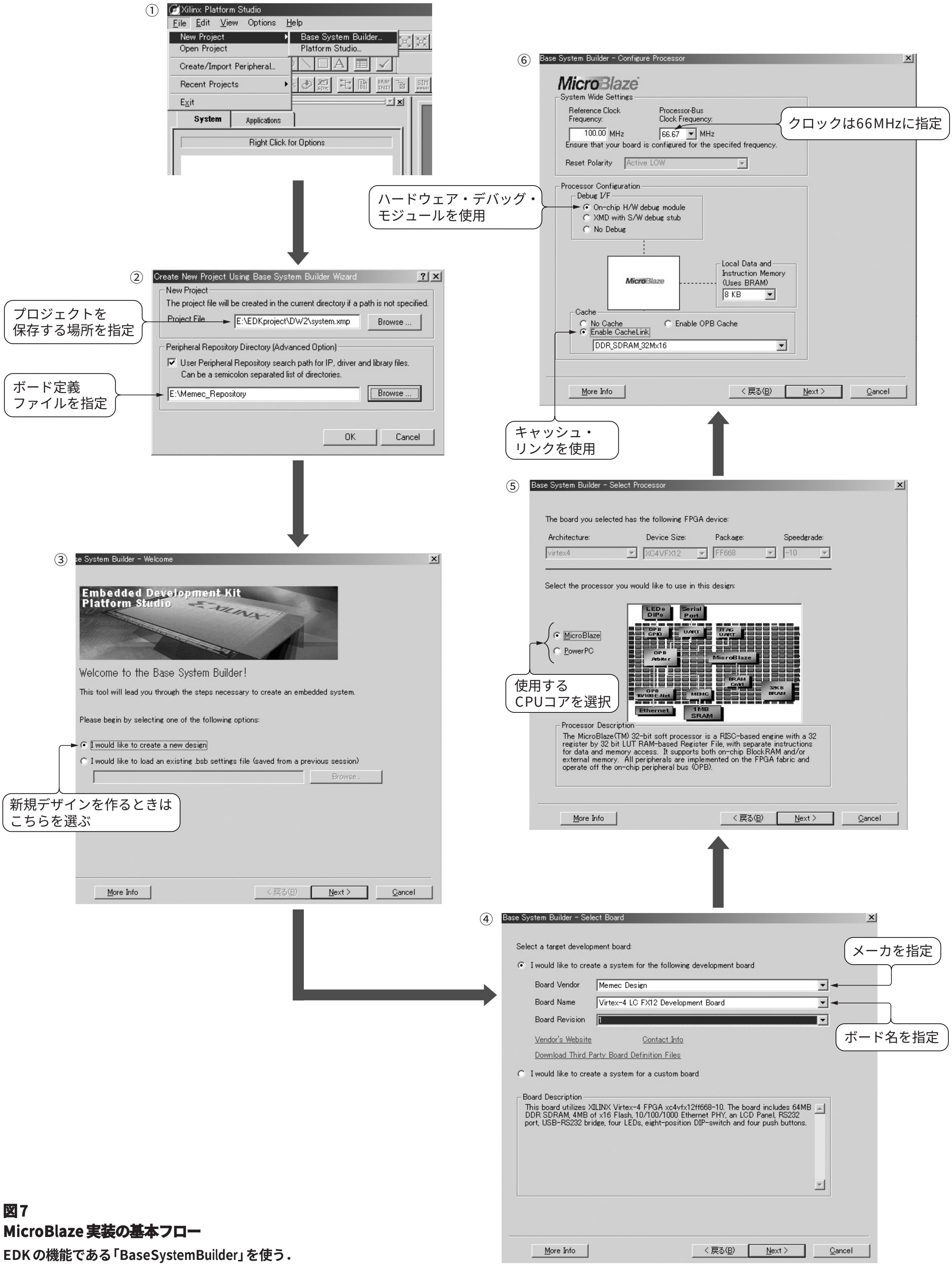
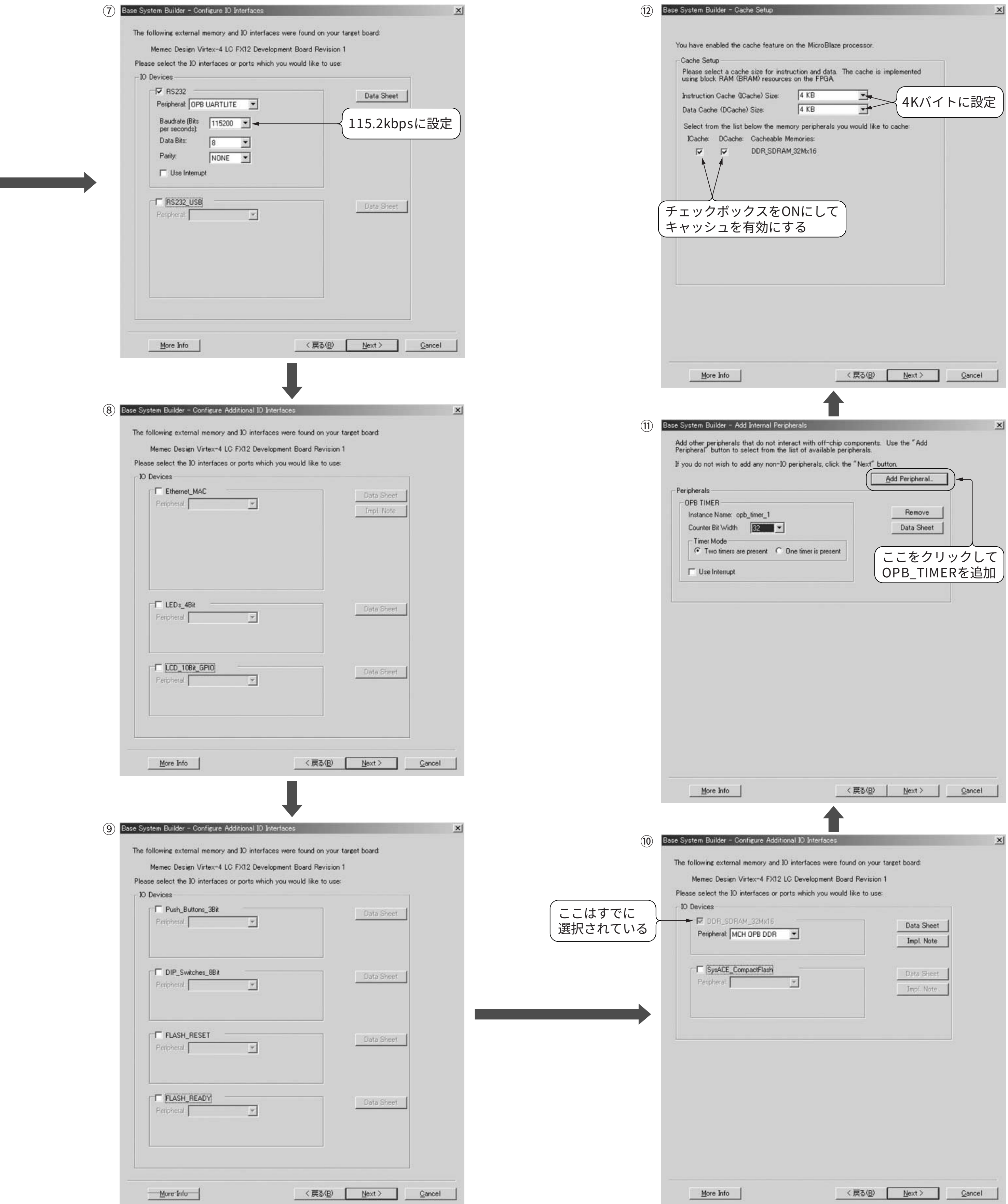


図7
MicroBlaze実装の基本フロー
EDKの機能である「BaseSystemBuilder」を使う．


⑦
Base System Builder - Configure IO Interfaces
The following external memory and IO interfaces were found on your target board:
Memec Design Virtex-4 LC FX12 Development Board Revision 1
Please select the IO interfaces or ports which you would like to use:
RS232
Peripheral: OPB UARTLITE
Baudrate (Bits per seconds): 115200
Data Bits: 8
Parity: NONE
Use Interrupt
RS232_USB
115.2kbpsに設定
⑧
Base System Builder - Configure Additional IO Interfaces
Ethernet_MAC
LEDs_4Bit
LCD_10Bit_GPIO
⑨
Base System Builder - Configure Additional IO Interfaces
Push_Buttons_3Bit
DIP_Switches_8Bit
FLASH_RESET
FLASH_READY
⑩
Base System Builder - Configure Additional IO Interfaces
Memec Design Virtex-4 FX12 LC Development Board Revision 1
DDR_SDRAM_32Mx16
Peripheral: MCH OPB DDR
SysACE_CompactFlash
ここはすでに
選択されている
⑪
Base System Builder - Add Internal Peripherals
Add other peripherals that do not interact with off-chip components. Use the "Add Peripheral" button to select from the list of available peripherals.
If you do not wish to add any non-IO peripherals, click the "Next" button.
Add Peripheral...
OPB TIMER
Instance Name: opb_timer_1
Counter Bit Width: 32
Timer Mode
Two timers are present
One timer is present
Use Interrupt
ここをクリックして
OPB_TIMERを追加
⑫
Base System Builder - Cache Setup
You have enabled the cache feature on the MicroBlaze processor.
Cache Setup
Please select a cache size for instruction and data. The cache is implemented using block RAM (BRAM) resources on the FPGA.
Instruction Cache (ICache) Size: 4 KB
Data Cache (DCache) Size: 4 KB
Select from the list below the memory peripherals you would like to cache:
ICache: DCache: Cacheable Memories:
DDR_SDRAM_32Mx16
4Kバイトに設定
チェックボックスをONにして
キャッシュを有効にする
More Info
< 戻る(B)
Next >
Cancel

```
BEGIN`opb_v20
`PARAMETER`INSTANCE`=`mb_opb
`PARAMETER`HW_VER`=`1.10.c
`PARAMETER`C_EXT_RESET_HIGH`=`0
`PORT`SYS_Rst`=`sys_rst_s
`PORT`OPB_Clk`=`sys_clk_s
END

BEGIN`opb_mdm
`PARAMETER`INSTANCE`=`debug_module
`PARAMETER`HW_VER`=`2.00.a
`PARAMETER`C_MB_DBG_PORTS`=`2
`PARAMETER`C_USE_UART`=`1
`PARAMETER`C_UART_WIDTH`=`8
`PARAMETER`C_BASEADDR`=`0x41400000
`PARAMETER`C_HIGHADDR`=`0x4140ffff
`BUS_INTERFACE`SOPB`=`mb_opb
`PORT`OPB_Clk`=`sys_clk_s
`PORT`DBG_CAPTURE_0`=`DBG_CAPTURE_s
`PORT`DBG_CLK_0`=`DBG_CLK_s
`PORT`DBG_REG_EN_0`=`DBG_REG_EN_s
`PORT`DBG_TDI_0`=`DBG_TDI_s
`PORT`DBG_TDO_0`=`DBG_TDO_s
`PORT`DBG_UPDATE_0`=`DBG_UPDATE_s
`PORT`DBG_CAPTURE_1`=`DBG_CAPTURE_2
`PORT`DBG_CLK_1`=`DBG_CLK_2
`PORT`DBG_REG_EN_1`=`DBG_REG_EN_2
`PORT`DBG_TDI_1`=`DBG_TDI_2
`PORT`DBG_TDO_1`=`DBG_TDO_2
`PORT`DBG_UPDATE_1`=`DBG_UPDATE_2
END

BEGIN`lmb_v10
`PARAMETER`INSTANCE`=`ilmb
`PARAMETER`HW_VER`=`1.00.a
```


⋮

```
`PORT`TX`=`fpga_0_RS232_TX
END

BEGIN`mch_opb_ddr
`PARAMETER`INSTANCE`=`DDR_SDRAM_32Mx16
`PARAMETER`HW_VER`=`1.00.a
`PARAMETER`C_MCH_OPB_CLK_PERIOD_PS`=`14999
`PARAMETER`C_REG_DIMM`=`0
`PARAMETER`C_DDR_TMRD`=`15000
`PARAMETER`C_DDR_TWR`=`15000
`PARAMETER`C_DDR_TWTR`=`1
`PARAMETER`C_DDR_TRAS`=`40000
`PARAMETER`C_DDR_TRC`=`65000
`PARAMETER`C_DDR_TRFC`=`75000
`PARAMETER`C_DDR_TRCD`=`20000
`PARAMETER`C_DDR_TRRD`=`15000
`PARAMETER`C_DDR_TRP`=`20000
`PARAMETER`C_DDR_TREFI`=`7800000
`PARAMETER`C_DDR_DWIDTH`=`16
`PARAMETER`C_DDR_AWIDTH`=`13
`PARAMETER`C_DDR_COL_AWIDTH`=`10
`PARAMETER`C_DDR_BANK_AWIDTH`=`2
`PARAMETER`C_MEM0_BASEADDR`=`0x24000000
`PARAMETER`C_MEM0_HIGHADDR`=`0x27ffffff
`PARAMETER`C_NUM_CHANNELS`=`4
`BUS_INTERFACE`SOPB`=`mb_opb
`BUS_INTERFACE`MCH0`=`ixcl
`BUS_INTERFACE`MCH1`=`dxcl
`BUS_INTERFACE`MCH2`=`ixcl2
`BUS_INTERFACE`MCH3`=`dxcl2
`PORT`DDR_Clk`=`fpga_0_DDR_SDRAM_32Mx16_DDR_Clk
`PORT`DDR_Clkn`=`fpga_0_DDR_SDRAM_32Mx16_DDR_Clkn
`PORT`DDR_Addr`=`fpga_0_DDR_SDRAM_32Mx16_DDR_Addr
`PORT`DDR_BankAddr`=`fpga_0_DDR_SDRAM_32Mx16_DDR_BankAddr
`PORT`DDR_CASn`=`fpga_0_DDR_SDRAM_32Mx16_DDR_CASn
`PORT`DDR_CKE`=`fpga_0_DDR_SDRAM_32Mx16_DDR_CKE
```


⋮

```


BEGIN`microblaze
`PARAMETER`INSTANCE`=`microblaze_1
`PARAMETER`HW_VER`=`4.00.a
`PARAMETER`C_DEBUG_ENABLED`=`1
`PARAMETER`C_NUMBER_OF_PC_BRK`=`2
`PARAMETER`C_NUMBER_OF_RD_ADDR_BRK`=`1
`PARAMETER`C_NUMBER_OF_WR_ADDR_BRK`=`1
`PARAMETER`C_USE_ICACHE`=`1
`PARAMETER`C_CACHE_BYTE_SIZE`=`4096
`PARAMETER`C_USE_DCACHE`=`1
`PARAMETER`C_DCACHE_BYTE_SIZE`=`4096
`PARAMETER`C_ICACHE_USE_FSL`=`1
`PARAMETER`C_DCACHE_USE_FSL`=`1
`PARAMETER`C_ICACHE_BASEADDR`=`0x24000000
`PARAMETER`C_ICACHE_HIGHADDR`=`0x27ffffff
`PARAMETER`C_ADDR_TAG_BITS`=`14
`PARAMETER`C_DCACHE_BASEADDR`=`0x24000000
`PARAMETER`C_DCACHE_HIGHADDR`=`0x27ffffff
`PARAMETER`C_DCACHE_ADDR_TAG`=`14
`BUS_INTERFACE`DOPB`=`mb_opb
`BUS_INTERFACE`IOPB`=`mb_opb
`BUS_INTERFACE`IXCL`=`ixcl2
`BUS_INTERFACE`DXCL`=`dxcl2
`PORT`CLK`=`sys_clk_s
`PORT`DBG_CAPTURE`=`DBG_CAPTURE_2
`PORT`DBG_CLK`=`DBG_CLK_2
`PORT`DBG_REG_EN`=`DBG_REG_EN_2
`PORT`DBG_TDI`=`DBG_TDI_2
`PORT`DBG_TDO`=`DBG_TDO_2
`PORT`DBG_UPDATE`=`DBG_UPDATE_2
END
```


図8　CPUコアの追加方法
MHSファイルを編集する方法を示す．GUIベースで追加する方法もある．

まず，テキスト・エディタでMHSファイルを開き，MicroBlaze_0の記述部分をそのままMHSファイルの最後にコピー&ペーストします．このままでは，同じインスタンス名になってしまうので，インスタンス名を変更します．これは，好きな名まえを付けることができます．キャッシュ・リンクやデバッグ・ポートは，1対1の接続のポートなので，これも名まえを変更しておく必要があります．追加したMicroBlazeにはBlockRAMは接続しないので，LMB接続部分（lmb）は削除します（**図8**）．

今回，デバッグ・ポート経由でメモリにソフトウェアをダウンロードするため，MicroBlaze_1もデバッグ・モジュールに接続しておかなければなりません．そこで，デバッグ・モジュールに接続するために，デバッグ・モジュール部分の接続ポート数の設定（C_MB_DBG_PORTS）を1から2に変更し，また，ポートの2番目にMicroBlaze_1の信号を接続するため，**図8**のいちばん上の図のように書き換えます．このとき，先ほどMicroBlaze_1で変更したデバッグ・ポートの信号名と一致するようにします．

さらに，今回はキャッシュ・リンクを使うため，キャッシュ・リンク・ポートをDDR SDRAMに接続しなければなりません．そこで，DDR SDRAMのキャッシュ・リンク・ポート数（C_NUM_CHANNELS）をデフォルトから4に変更し，また，キャッシュ・リンク・ポートの3番目と4番目に，MicroBlaze_1の命令とデータのキャッシュ・リンク・ポートを接続します．このときも，MicroBlaze_1のキャッシュ・リンクの信号名と名まえが一致するようにします．

以上でハードウェアは完成です．なお，この追加によるリソースの増加量は，787スライス（約1600ロジック・セル）でした．

● ソフトウェアの実装

次に，ソフトウェアの実装に入ります．今回はインターネットでソース・コードが公開されていたJPEGのエンコード/デコード・ソフトウェア「jpegcs/jpegls」を使いました（http://nikq.nothing.sh/works.html）．速度最適化はさほど行われていないようですが，小型軽量なので，組み込み用途には向いているかと思います．

今回の評価では，これをMicroBlazeで動作させるために一部書き換えを行いました．

● 性能の測定

性能を評価するために，まずは一つのCPUコアで動かしてみます．352×288ピクセル，24ビット・カラーの画像をエンコードさせてみると，1フレーム当たりの処理時間は842msとなりました．続いて，この圧縮されたJPEGデータをデコードすると，処理時間は803msでした．これらの処理を，一つのCPUコアで構成されたシステムで実行すると，二つの合計である1.6s以上かかる計算になります．実際には，これにタスク切り替えなどのオーバヘッドが加わることになります．

次に二つのCPUコアを使って動作させてみます．MicroBlaze_0にエンコードの処理を，MiroBlaze_1にデコードの処理を割り当てて，繰り返し実行させます．すると，エンコードの平均処理時間は877ms，デコードの平均処理時間は889msとなりました．1CPUのときと比べて時間がかかるのは，共有メモリを使ったシステム構成のため，バスの競合が起こっているものと考えられます． エンコードに比べて，デコードのほうが性能低下が大きいのは，MicroBlaze_0のほうがバスを優先して使えるためです．1CPUのときと比べてそれほど大きな性能低下にならずにすんでいるのは，キャッシュ・メモリの効果が高いためです．2個のCPUを同時に動作させることにより，2倍近いシステム性能が得られることになります．

とりあえずこれで，バスを共有して2個のMicroBlazeを実行させることの有効性を確認できたかと思います．

● CPUコアをさらに追加する

2個のMicroBlazeの場合，性能の低下があまり見られなかったので，共有バスにはさらにMicroBlazeを追加するだけのゆとりがありそうです．そこで，もっとCPUコアを増やすことを試してみます．

DDR SDRAMコントローラにはキャッシュ・リンク用のポートは四つしかなく，二つのMicroBlazeの命令キャッシュ，データ・キャッシュそれぞれに接続しているので，すでに使い切ってしまっています．そこで，三つ目のMicroBlazeはキャッシュ・リンクを使わず，普通のキャッシュを使ってみました．そして三つのCPUを同時に動かしたところ，キャッシュ・リンクを使っている二つのMicroBlazeは900ms弱の処理時間で実行できたのですが，三つ目のMicroBlazeによるデコードは1.8s前後もかかってしまいました．DDR SDRAMなどの比較的レイテンシの大きいメモリは，キャッシュ・リンクを使わないとキャッシュの効果がうまく発揮できないようです．

そこで，MicroBlaze_0では，ローカル・メモリにソフトウェアを入れて実行し，MicroBlaze_1と新たに追加したMicroBlaze_2にキャッシュ・リンクを使うことにしてみました．すると，MicroBlaze_1とMicroBlaze_2は900ms前後の処理時間だったのに対し，MicroBlaze_0はバスの使用権が高いにもかかわらず，1.3s弱の処理速度になりました．しかし，前回と比べると性能の低下は小さく，まだ実用になりそうです．もちろん，作業用データもローカル・メモリに収めることができれば共有メモリを使わずにすむので，性能も確保できます．

● 音声処理部を追加

先ほどの結果から，共有メモリ上にソフトウェアを置く場合は二つまでが限界で，それ以上はローカル・メモリ上に置く必要があるようです．音声圧縮/伸張プログラムを小型で軽量に作ることができれば，ローカル・メモリに格納することにします．

マイクとスピーカは，EDKにあらかじめ用意されているΔ-Σ型D-A/A-DコンバータのIPコアを使うと簡単に実現できます．これらのIPコアはFIFO（first-in first-out）メモリを持っており，FIFOメモリのフル/エンプティなどは割り込みでプロセッサに通知します．CPUの割り込み応答速度も重要になりそうですが，ローカル・メモリ上でソフトウェアが動作しているのであれば，バスの競合の影響を受けずにすみます．また，1CPU当たり一つの割り込みになり，多重割り込みの可能性がないため，十分な性能を確保できます．

テレビ電話のようなシステムの場合，画像のほうは少々の遅延が許容されるため，共有メモリ上で動かすとよいかと思います．

ディスプレイにはCRTコントローラを用意しなければなりませんが，カメラにはFSLを使用すると便利でしょう．画像サイズが352×288ピクセル，16ビットYUVの解像度で，30フレーム/sでデータが出力されるカメラをFSLに接続したことがありますが，データ・ポートをそのまま直結するだけで，MicroBlazeのFSLリード命令できちんとデータが受信できました．

これらの構成をブロック図にすると，**図9**のようになります．割り込みが1CPU当たり一つなので，割り込みコン

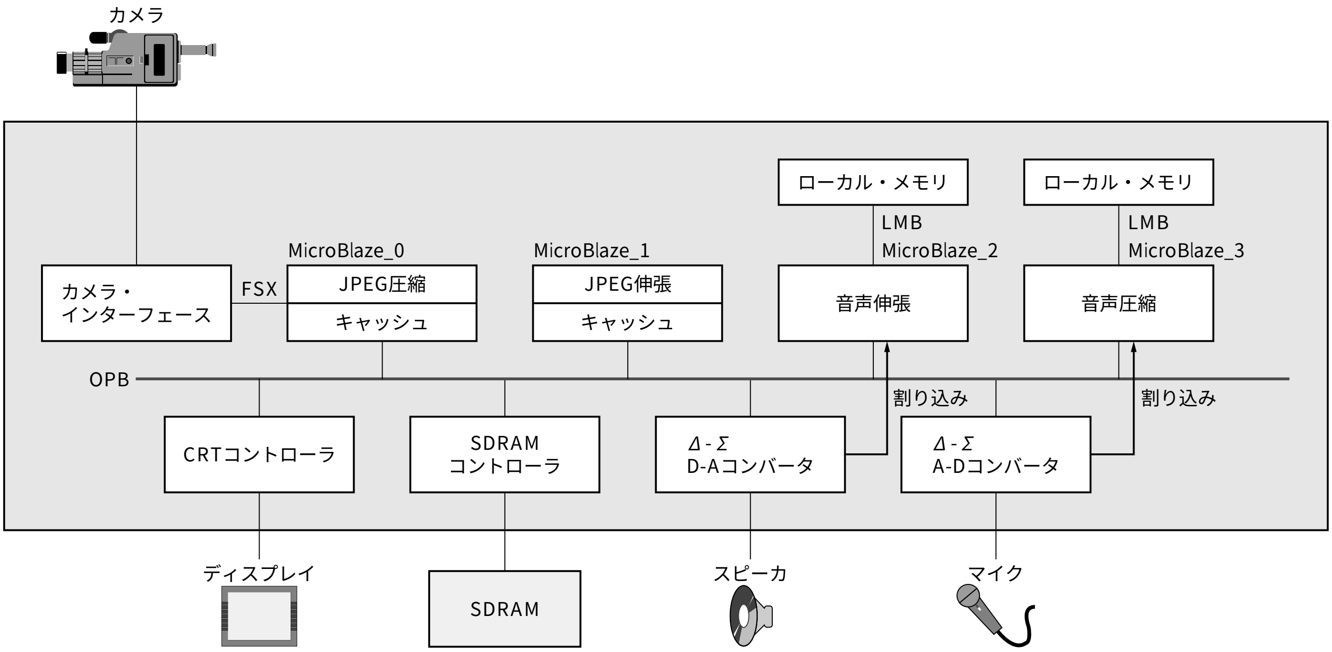


図9　テレビ電話端末のハードウェア・ブロック図

トローラは必要ありません．

CRTコントローラ，カメラ・インターフェース部分を除いて実際にこのシステムを構成したところ，3763スライス，16 BlockRAMとなりました．これは，Spartan-3EファミリのXC3S500E（4656スライス，20 BlockRAM）に実装できる規模です．さらに900スライス，4 BlockRAMが残るので，ほかに必要なI/Oなどの回路に使えます．

この構成をとると，ほとんどの機能を1チップに収められるため，部品点数は最小になり，かなり小型の装置を実現できそうです．

4. まとめ

以上，同じバスに複数のCPUコアを接続し，消費リソースを抑えながら，性能を向上させる例を紹介しました．

今回は，機能分散という形でしたが，JPEG圧縮速度をもっと上げたい場合は，負荷分散という形の並列処理になります．その場合，ソフトウェア側の実装をいろいろとくふうする必要があると思いますが，ハードウェアについては同じような構成で，同程度の効果が得られると思います．

一つのシステムにいろいろな機能を持たせたい場合で，一つのCPUだけでは処理が間に合わないときは，複数個のCPUによる並列処理という方法もなかなかおもしろいと思います．

参考・引用*文献

(1) 澤田修，森山俊彦；FPGA向けマイクロプロセッサのアーキテクチャ ── Xilinx社の32ビットRISC CPUコア「MicroBlaze」，pp.76-83，Design Wave Magazine，2002年2月．
(2) 山口耕作，濱石悟，田中良平；FPGA向けマイクロプロセッサを評価する ── MicroBlazeでつくるディジタル時計，pp.84-95，Design Wave Magazine，2002年2月．
(3) 山口耕作；徹底比較！ ハード・マクロvs. ソフト・マクロ ── Virtex-II ProとMicroBlazeの活用法，pp.22-31，Design Wave Magazine，2004年12月．

おおまき・まさとも
メメックジャパン（株）

<筆者プロフィール>
大牧正知．小学校低学年の頃，父親が持っていた「初歩のラジオ」という雑誌がきっかけで電子工作への道を歩む．それ以来20年余り，あれこれやってきたが，今だにわからないことだらけである．